Quantum Floyd-Warshall Algorithm

Abhishek Santosh Gupta and Anirban Pathak

1st February 2008

Jaypee Institute of Information Technology, A-10, Sector-62, Noida-201307, UP, India

Abstract

Classical Floyd-Warshall algorithm is used to solve all-pairs shortest path problem on a directed graph. The classical algorithm runs in $\mathcal{O}(V^3)$ time where V represents the number of nodes. Here we have modified the algorithm and proposed a quantum algorithm analogous to Floyd-Warshall algorithm which exploits the superposition principle and runs in $\mathcal{O}(Vlog_2V)$time.

As silicon based technology is approaching saturation, the alternative proposals like DNA computing and quantum computing are drawing more and more attention of the scientific community. A considerable amount of research on quantum computer is already done [1-12 and references there in]. We have very nice proposals for construction of quantum gates [2-3], implementation of quantum cryptography [4-5], quantum teleportation [6] etc. In 1994, Peter Shor discovered a polynomial time quantum algorithm for factorisation [9]. Just after two years, Lov Grover discovered a search algorithm which is quadratically better than conventional search. We have some other quantum algorithms [11 and references there in] which are supposed to run faster than their classical counter part. But basically, in a broad sense there exist three different quantum algorithms e.g. Deutsch-Josza algorithm [10] (which is a generalization of the Deutsch algorithm) to find out whether a function is constant or balanced, Shor's algorithm [9] and Grover's algorithm [8]. All other existing algorithms are some way or other variation of these algorithms. Recently Peter Shor has nicely explained the reason why the number of quantum algorithms is so small [12]. Keeping this in view, here we have tried to propose a new quantum counter part of modified classical Floyd-Warshall algorithm.

Grover's algorithm and Shor's algorithm have got much more attention than the DJ algorithm because of their potential applications in fast database search and breaking of RSA cryptographic code by fast factorization. Therefore, any quantum algorithm which runs faster than classical counter part and which have

potential application in real life is important and interesting. It is an interesting curiosity to note that all these algorithms essentially exploit superposition principle to establish its advantage over their classical counter part. In the present work we have also exploited superposition principle to write a quantum version of the modified classical Floyd-Warshall algorithm. The classical Floyd-Warshall (FW) algorithm is used to find out the shortest path between any two nodes in the graph G having V nodes[13]. In the present work we have modified the existing classical algorithm and have generalized that for a quantum implementation. The classical algorithm runs in $\mathcal{O}(V^3)$ time whereas the proposed quantum algorithm runs in $\mathcal{O}(Vlog_2V)$time. Thus the advantage of quantum algorithm increases with the increase in number of nodes V. In the next section we will describe classical Floyd-Warshall algorithm and its modification. In section 2 we discus the quantum FW algorithm and section 3 is dedicated to complexity analysis of the proposed algorithm and conclusions.

1 The classical Floyd-Warshall algorithm

Let G be a graph with V nodes and E edges. The classical FW algorithm is used to find out the shortest path between any two nodes in the graph. The weight on each edge can represent any function that we are interested in minimizing (or maximizing). One of the best real life examples of FW algorithm is the situation in which one has to find out least fare between any 2 cities and one can take as many stoppages as he wants. A C code for the classical FW algorithm is given below to show that classically we need three 'for loops' varying from 0 to V-1 in steps of 1, in order to minimize the function of our interest and that leads to a time complexity $\mathcal{O}(V^3)$

```
// post initialization of dist[][]
for( k=0;k<V;k++)
 {
  for(i=0;i<V;i++)
   {
    for(j=0;j<V;j++)
     {
       if ( dist [i][j] > dist[i][k] + dist[k][j] )
       {
       dist[i][j]=dist[i][k]+dist[k][j] ;
       pred[i][j]=k ;
       }
     }
   }
 }
```

1.1 Key Observations

1. **Observation 1:** If dist[i][j] is assumed to be positive then whenever dist[i][j] is modified/updated to a new (lesser) value all the indices i,j,k will be different or in other words no 2 of i,j,k will be same when an update on dist[i][j] will be taken place[1]
   Proof:
   Case 1: dist[i][j] is modified only if RHS of below equation has lesser value than current LHS value

   $$dist[i][j] = dist[i][k] + dist[k][j]$$

   If k=i then we will get

   $$dist[i][j] = dist[i][i] + dist[i][j]$$

   assuming dist[i][i]>=0 and thus no update/modification is possible. Similarly we can reach to the same conclusion for the other cases (i.e., k=j follows similarly and i=j is trivial).

2. **Observation 2:** For a particular value of k, once dist[i][j] is modified/updated it will never appear on the right hand side of

   $$dist[i][j] = dist[i][k] + dist[k][j]$$

   for the same value of k.
   Proof by reductio ad absurdum:
   for k=k1,suppose dist[i1][j1] is modified i.e i=i1 and j=j1 and dist[i1][j1] appears on right hand side of expression : dist[i][j]=dist[i][k]+dist[k][j] for same value of k(k=k1)
   Case 1: if dist[i][k]=dist[i][j1]
   giving k=j1 [but this is not possible as k=k1 and according to the observation 1 $k1 \neq j1$]
   Case 2: if dist[k][j]=dist[i1][j1]
   giving k=i1[but this is not possible as k=k1 and as per observation (1) $i1 \neq j1$]

1.2 The Modified FW Algorithm

The second observation can be exploited to construct a modified FW algorithm. The modified algorithm is such that it can easily be implemented in the quantum domain and by using the advantage of superposition principle the run-time complexity of the algorithm can be improved.

For each value of k, we will construct a new dist[i][j] (for all i & j), i.e. dist[i][j] will now become 3 dimensional with new dimension along k as dist[k][i][j].

[1] Intrinsic assumption is that dist[i][j] is always positive(>=0). This means negative weightage is not included. There exist analogous classical algorithm which consider negative weights.

Ideally, only dist[k-1][i][j] is required to update current copy dist[k][i][j]. Thus, the third dimension requires only 2 values. The (k mod 2) function will help us to alternately access either dist[0][i][j] or dist[1][i][j] with either one alternating between old and current dist[i][j]. This is the essential idea behind our modified algorithm which can be written in a C program as

1.2.1 C code for the Modified FW Algorithm

```
// post initialization of dist[][]
   for (k=0; k<n; k++)
     {
      for (i=0; i<n; i++)
       {
        for (j=0; j<n; j++)
         {
dist[(k+1)%2][i][j]=MIN(dist[(k+1)%2][i][j],(dist[k%2][i][k]+dist[k%2][k][j]));
         }
       }
     }
```

The above modified code is tested for various datasets and it is found to produce correct results. The advantage of the modified code over the existing code is that it can be exploited to take the advantage of superposition principle while we write a quantum counter part of the same

2 The Quantum FW Algorithm

The basic trick of the proposed quantum algorithm is the observation that a for loop can be replaced by a suitable number of Hadamard gates with necessary number of qubits. For example, we can use n Hadamard gates operating on n input qubits to produce an equal superposition of 2^n states and this superposition of states can replace a for loop which runs from i=0 to i=2^n in steps of 1 [$for\,(i = 0;\, i < V;\, i++),\, V = 2^n$].

There exist a major difference between a classical for loop and a quantum superposition. The loop is sequential whereas the Hadamard superposition is parallel in nature. Therefore, we can not replace those for loops for which it is essential to execute the statements inside the loop in sequence only. But the other loops can be replaced. Notice that in the modified FW algorithm, the innermost loops of i & j can be replaced by Hadamard gates with appropriate input qubits as parallel execution does not affect the logical end result.

The quantum FW algorithm works in following steps:

1. A n bit register contains all qubits in state $\mid 0 >$ (This register represents k)

2. The algorithm starts with 2^n qubits in state $|0 >$. They are subdivided into two sets of n qubit each prepared in state $|0 >$ (i.e. we are starting

with $|0>^{\otimes n} \otimes |0>^{\otimes n}$) and Hadamard transform is used to put the computer in the following equal superposition state

$$|\Psi> = |i> \otimes |j> = \frac{1}{\sqrt{n}} \Sigma_{x=0}^{n-1} |x> \otimes \frac{1}{\sqrt{n}} \Sigma_{y=0}^{n-1} |y>$$

3. Now the ORACLE is queried to get :
   d0 = dist [(k+1) mod 2][| i >][| j >]
   d1= dist [k mod 2][| i >][k]
   d2= dist [k mod 2][k][| j >]
   dist[][][] denotes the function that we wish to minimize. It is 3 dimensional as argued above and d0, d1, d2 denote qubit registers used to store query results. Their size depends upon the function that we wish to minimize (or maximize). d0,d1,d2,d3(later) should be large enough to store the maximum possible sum of all function values.

4. d3 is computed via d3=d1+d2

5. IF d3 $<$ d0 then goto step 6 else goto step 7

6. Update
   dist [k mod 2][| i >][| j >] = d3
   pred [| i >][| j >] = k
   pred[|i>][|>j] is used to store the node that connects the shortest path between i & j so that this information can be used to reconstruct the shortest path afterwards. pred[i][j] does not require a third dimension since dist[k][i][j] is modified only once for each value of k for a particular i and j pair.

7. IF k = V -1 STOP else increment k and goto step 2.

3 Conclusions

The outer loop on k has complexity $\mathcal{O}(V)$. The inner Hadamard gate operations to create superposition has complexity $\mathcal{O}(n)$where $n = log_2(V)$. Thus, the overall complexity of algorithm is $\mathcal{O}(Vlog_2V)$ as opposed to classical complexity of $\mathcal{O}(V^3)$. From the present complexity analysis it is very clear that when the number of node V is large then quantum FW algorithm will be more faster than its classical algorithm. Like Shor's algorithm [9] and Grover algorithm [8] the present algorithm do have applicability in real life and that's why its very important. Its importance also lies in the fact that the FW algorithm uses dynamic programming technique to find the shortest path between any two nodes in a graphical network. The proposed quantum algorithm can therefore be applied probably to many other dynamic programming problems with minor modifications.

References

1. D. Suter and J. Stolze, Quantum Computing, (Wiley, Weinheim) 2004
2. Z. Zhao *et al*, Phys. Rev. Lett, **94** (2005) 030501.
3. Z. I. Cirac and P. Zoler, Phys. Rev. Lett. 74 (1995) 4091.
4. N. Gisin *et al*, Rev. Mod. Phys. **74** (2002) 147.
5. Y. Nambu and H. Kosaka, NEC Res. Develop. 44 (2003) 285.
6. A. Dantan *et al*, Phys. Rev. Lett **94** (2005) 050502.
7. L. Fortnow, NEC Res. Develop. 44 (2003) 269.
8. L. K. Grover, Phys. Rev. Lett. **78** (1997) 325.
9. P.W. Shor, Proc. 35th IEEE (IEEE press, Los Alamitos CA, 1994), p. 352; quant-ph/9508027.
10. M. A. Nielsen and I. A. Chuang, Quantum computation and quantum information (Cambridge University Press, New Delhi), 2002.
11. A. Ambainis, Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, **111** (2004) 5.
12. P. W. Shor, J A C M, **50** (2003) 87.
13. C. H. Thomas *et al*, Introduction to algorithm, (London MIT Press, Cambridge) 2001, p. 629.